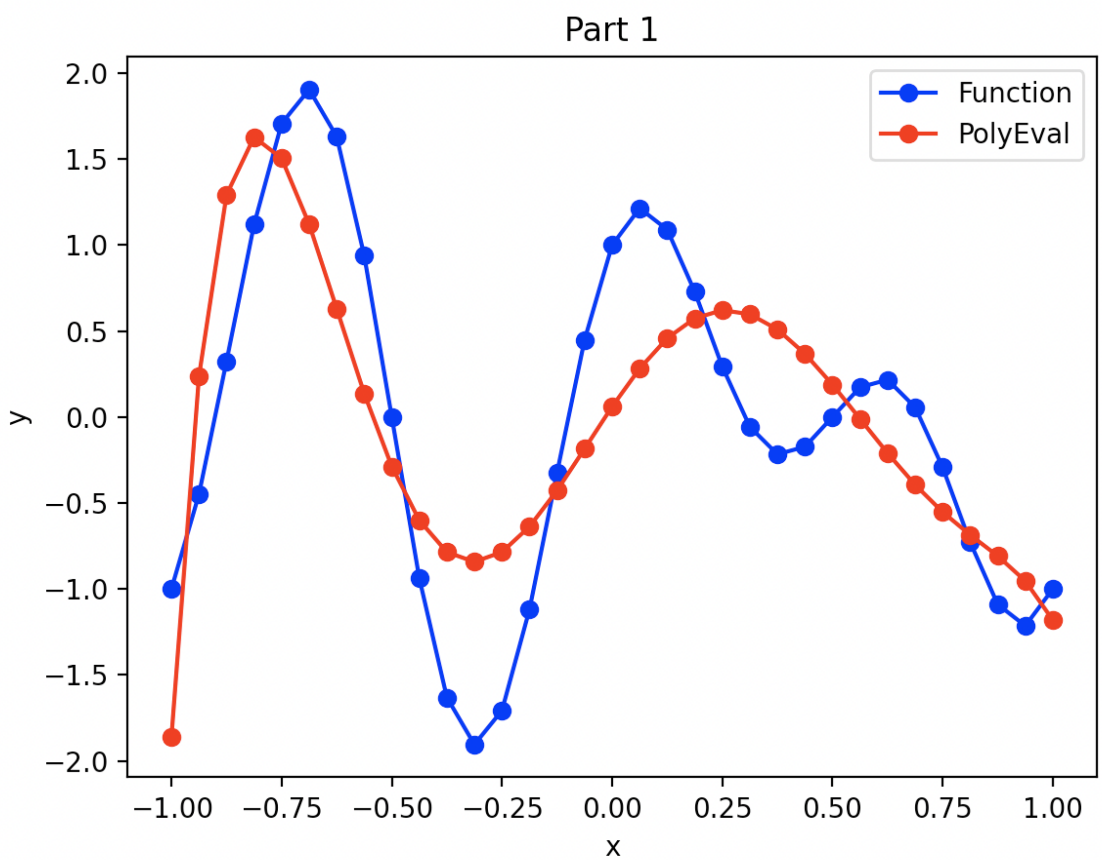
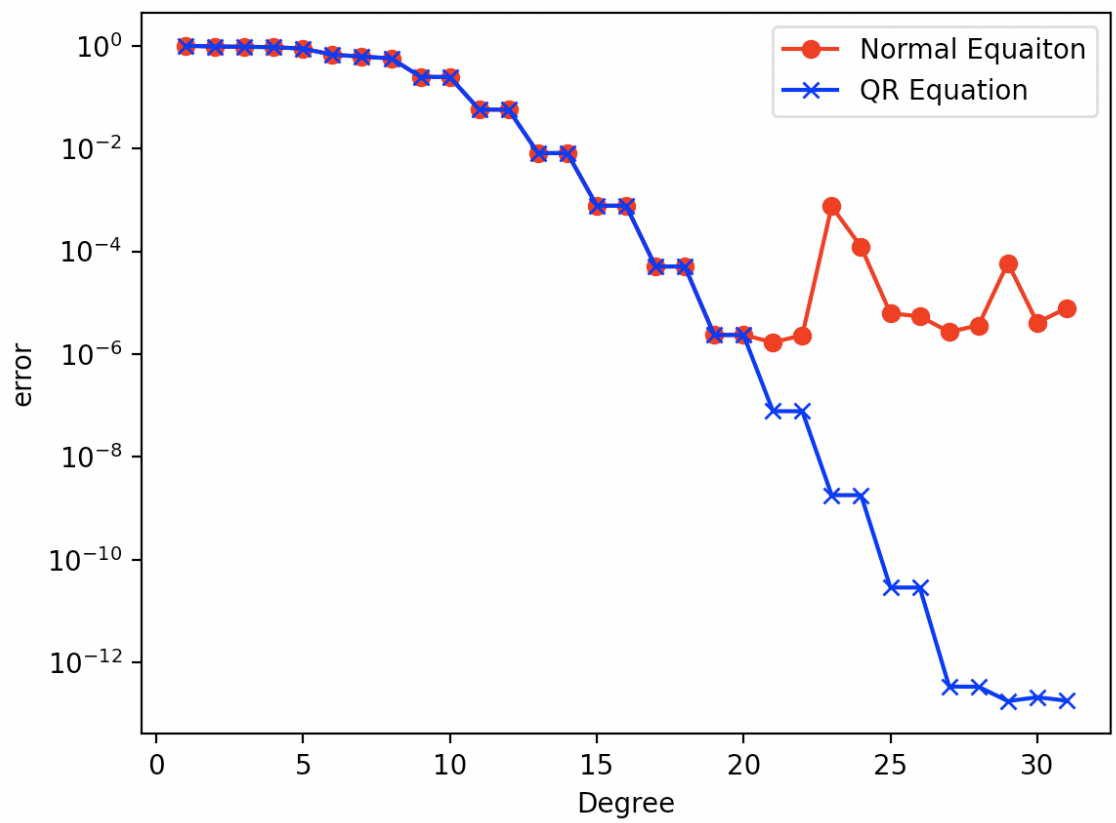
Numerical HW 9 - CSCI 3656 - 001
Kai Handelman
11/07/21

1.



2.



As expected, as the number of degrees in the polynomial increases the error for both methods decreased when compared to the actual function. We can also see the normal equation start to struggle later on due to how the method has an increased condition number (power of 2) compared to the QR equation.

Code:

```
import numpy as np
import numpy.linalg as ln
from numpy.linalg.linalg import norm
import numpy.random as rn
import matplotlib.pyplot as plt

def getTranspose(matrix):
    tMatrix = [[0 for j in range(len(matrix))] for i in range(len(matrix[0]))]
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            tMatrix[j][i] = matrix[i][j]
    return tMatrix

def backSub(matrix,right):
    x = np.zeros_like(right,float)
    for i in range(len(matrix)-1,-1,-1):
        temp = right[i][0]
        for j in range(len(matrix[i])-1, i, -1):
            temp = temp - matrix[i][j] * x[j]
        x[i] = temp / (matrix[i][i])
    return x

def thinQR(a,b):
    q,r = ln.qr(a,mode='reduced')
    nB = np.dot(getTranspose(q), b)
    x = backSub(r,nB)
    return x

def normEq(a,b):
    aTa = np.dot(getTranspose(a),a)
    atB = np.dot(getTranspose(a),b)
    # try:
    #     r = ln.cholesky(aTa)
    # except:
    #     print("> Not positive definite\n\n\n\n\n")
    #     return None
    y = ln.solve(aTa,atB)
    return y# ln.solve(getTranspose(r),y)
```

```python
def errFunc(xPoints, fun, poly):
    tempT = 0
    tempB = 0
    for i in xPoints:
        tempT = tempT + np.power((fun(i) - np.polyval(poly,i)),2)
        tempB = tempB + np.power(fun(i),2)
    return np.sqrt(tempT/tempB)


def main():
    # Part 1
    xPoints = np.linspace(-1,1,33)
    func = lambda x: np.sin(2*np.pi*x)+ np.cos(3*np.pi*x)
    yPoints = func(xPoints)
    trueTable = [[t,func(t)] for t in xPoints]
    a = [[np.power(trueTable[i][0],t) for t in range(7)] for i in
range(len(trueTable))]
    b = [[t[1]] for t in trueTable]
    c = np.flip(thinQR(a,b))
    trainR = [np.polyval(c,i)for i in xPoints]

    # plt.xlabel("x")
    # plt.ylabel('y')
    # plt.plot(xPoints,yPoints,'bo-',label="Function")
    # plt.plot(xPoints,trainR,'ro-',label="PolyEval")
    # plt.legend()
    # plt.title("Part 1")
    # plt.show()

    # Part Two
    xTesting = rn.uniform(-1,1,100)
    yTesting = func(xTesting)
    testTable = [[t,func(t)] for t in xTesting]

    neE = []
    qrE = []
    for d in range(1,32):
```

```
        a = [[np.power(trueTable[i][0],t) for t in range(d)] for i in
range(len(trueTable))]
        b = [[t[1]] for t in trueTable]
        qrC = np.flip(thinQR(a,b))
        neC = np.flip(normEq(a,b))
        neE.append(errFunc(xPoints,func,neC))
        qrE.append(errFunc(xPoints,func,qrC))
        # qrE.append(ln.norm(yTesting-
np.polyval(qrC,xTesting))/ln.norm(yTesting))
        # neE.append(ln.norm(yTesting-
np.polyval(neC,xTesting))/ln.norm(yTesting))


    finalX = np.arange(1,32)
    plt.semilogy(finalX,neE,'ro-',label="Normal Equaiton")
    plt.semilogy(finalX,qrE,'bx-',label="QR Equation")
    plt.xlabel("Degree")
    plt.ylabel("error")
    plt.legend()
    plt.show()




main()
```